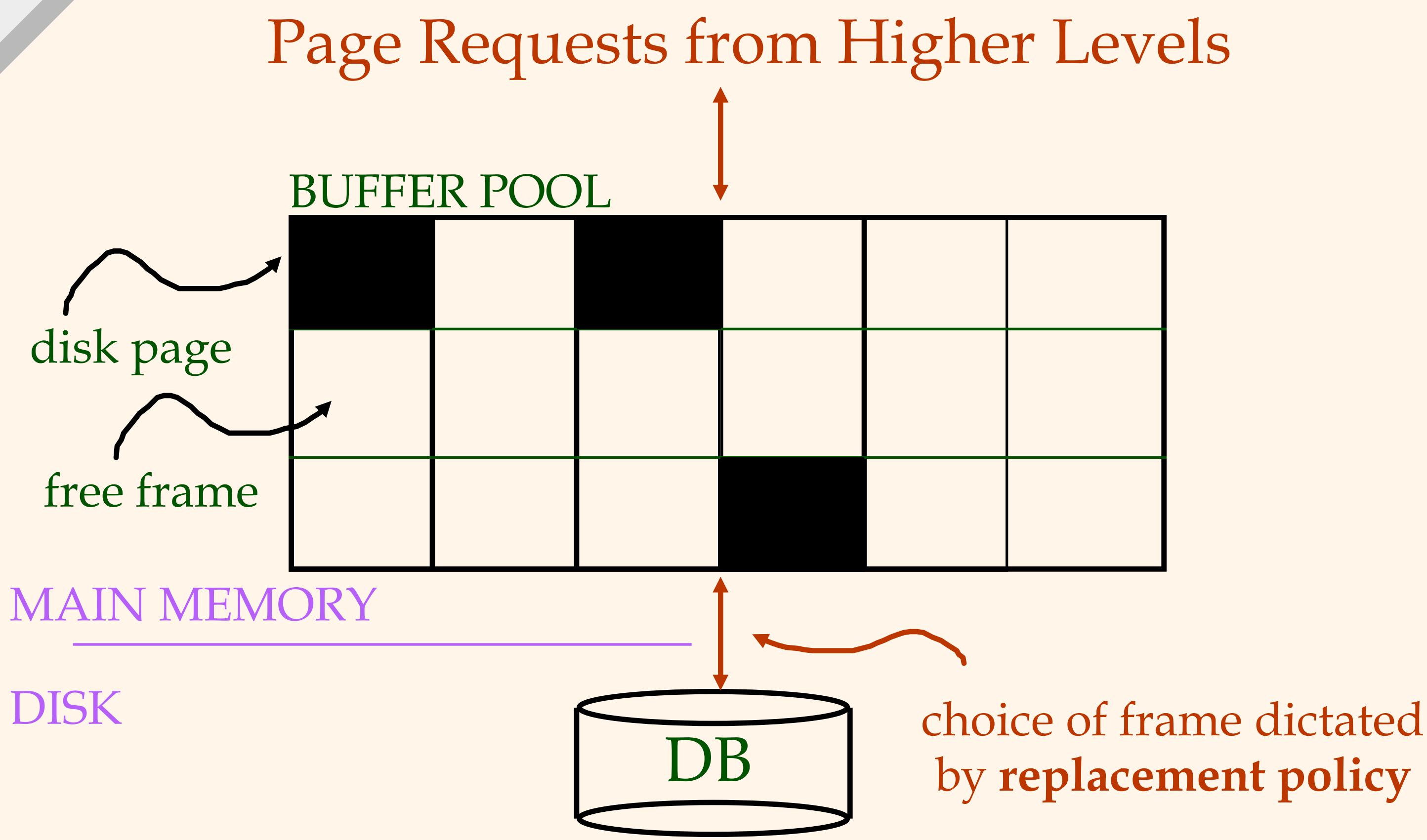
ARIES wrap-up

Last time

- Started on ARIES
- A recovery algorithm that guarantees Atomicity and Durability after a crash

Buffer Management in a DBMS



Handling the Buffer Pool

- Force every write to disk when a transaction commits?
  - If yes, poor response time.
- Steal buffer-pool frames from uncommited transactions?
  - If not, poor throughput.

	No Steal	Steal
Force		
No Force		Desired

ARIES metadata summary

LogRecords

prevLSN
transID
type
pageID
length
offset
before-image
after-image

Data pages

each
with a
pageLSN

master record

Transaction Table

lastLSN
status

Dirty Page Table

recLSN

flushedLSN

Checkpoints

- Periodically take a snapshot of the transaction table and dirty page table and write to log
- Don't snapshot the whole DB!!

Normal Execution of a Transaction

- Commit: write commit log record, flush log up to lastLSN, write end log record
- Abort: write abort log record, undo changes by "playing back" log in reverse order
  - Write a CLR (compensation log record) for each undo action
  - When done, write end log record

Example

- 10 T1 writes P3 (prevLSN: NULL)
- 20 T2 writes P4 (prevLSN: NULL)
- 30 T2 writes P5 (prevLSN: 20)

Frame steal - P4 gets written to disk by BM (log must be flushed up to 20)

T2 aborts

- 40 Abort T2
- 50 CLR T2 P5 (undoNextLSN = 20)

Frame steal - P5 gets written to disk by BM (log must be flushed up to 50)

- 60 CLR T2 P4 (undoNextLSN = NULL)
- 70 End T2
- 80 T1 commits

Flush log up to log record 90, then the commit(T1) returns

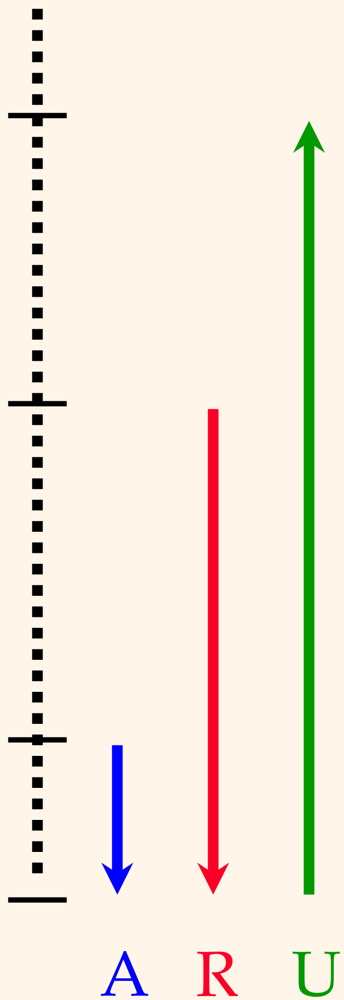
Crash Recovery: Big Picture

Oldest log rec. of transaction active at crash

Smallest recLSN in dirty page table after Analysis

Last chkpt

CRASH



- Start from a checkpoint (found via master record).
- Three phases. Need to:
  - Figure out which transactions committed since checkpoint, which failed (Analysis).
  - REDO ***all*** actions.
    - (repeat history)
  - UNDO effects of failed transactions.

Recovery: The Analysis Phase

- Determines point in the log where to start Redo
- Determines (superset of) pages in buffer pool that were dirty at time of crash
- Determines transactions that were active at time of crash and must be undone

Recovery: The Analysis Phase

- Reconstruct state (transaction table and dirty page table) at checkpoint.
  - via end_checkpoint record.
- Scan log forward from checkpoint.
  - End record: Remove trans. from trans. table.
  - Other records: Add trans. to trans.table, set lastLSN=LSN, change trans. status on commit.
  - Update record: If P not in Dirty Page Table,
    - Add P to D.P.T., set its recLSN=LSN.
- When done, have reconstructed trans. table and DPT as they were at time of crash

Example of Analysis

RAM

transaction Table

lastLSN

status

Dirty Page Table

recLSN

flushedLSN

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

Recovery: The REDO Phase

- We *repeat history* to reconstruct state at crash:
  - Reapply updates (even of aborted transactions!), redo CLRs.
- Scan forward from log rec containing smallest recLSN in D.P.T. For each CLR or update log record, REDO the action unless a special condition holds (will see these soon)
- To REDO an action:
  - Reapply logged action.
  - Set pageLSN to LSN. No additional logging!

Recovery: The REDO Phase

- We do NOT need to redo a logged action if one of the following 3 conditions holds:
  - Affected page is not in the Dirty Page Table (after analysis) or
  - Affected page is in D.P.T., but has recLSN > LSN, or
  - pageLSN (in DB) ≥ LSN.

Example for Condition 3

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

P3 was written to disk
with pageLSN = 20

so on recovery
LSN = pageLSN

Example for Condition 1

- Affected page is not in DPT after analysis
- Checkpoint at LSN 1000
- DPT at checkpoint contains a page P1 with recLSN 800
- Sometime between 800 and 1000, page P2 was updated, written to disk and removed from DPT
- Recovery starts at 800 and will encounter P2 updates, but those don't need to be redone

Recovery: The UNDO Phase

- Need to undo changes by the "loser" transactions
- In reverse order in which they were applied
- To achieve this, we follow the back pointers (prevLSN entries) in the update logs

Example (part way thru UNDO)

RAM

transaction Table
- lastLSN
- status

Dirty Page Table
- recLSN

flushedLSN

ToUndo

LSN	LOG
00,05	begin_checkpoint, end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40,45	CLR: Undo T1 LSN 10, T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART
70	CLR: Undo T2 LSN 60
80,85	CLR: Undo T3 LSN 50, T3 end

undonextLSN

Crashes mid-recovery

- ARIES is robust to crashes during recovery itself
- Always allows you to reconstruct a consistent DB state no matter when crash happened.

Crash during Recovery

- What happens if system crashes during Analysis?
  - Restart analysis phase
- Crash during REDO?
  - Restart recovery with analysis and redo
  - Some changes from the first REDO may now have made it to disk and don't need to be redone

Crash during UNDO

RAM

transaction Table

lastLSN

status

Dirty Page Table

recLSN

flushedLSN

ToUndo

LSN	LOG
00,05	begin_checkpoint, end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40,45	CLR: Undo T1 LSN 10, T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART
70	CLR: Undo T2 LSN 60
80,85	CLR: Undo T3 LSN 50, T3 end
	CRASH, RESTART
90	CLR: Undo T2 LSN 20, T2 end

ARIES summary

- Recovery algorithm that uses Write-Ahead Logging to guarantee atomicity and durability
- Analysis-Redo-Undo phases